ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 1, January 2017

C&K Metric for Object Oriented System: A Review

Ekta Nehra

Student, Dept. of CSE, OPJS University, Churu, Rajasthan, India

ABSTRACT: This research is focused on factors required to improve the reliability. Research has considered C&K metric for object oriented system. For the purpose of increasing software reliability, a realistic estimate of software maintenance effort is used. Time-consuming and expensive estimate of maintenance effort was made. It's utilized for soft computing methods like prioritization and effort calculation. Techniques for soft computing have been developed that may be used to estimate effort. Traditional methods have been addressed in the research, as well as client requirements, method coverage, and cost considerations. The current research has been discussed and its contribution highlighted. The goal of the project is to examine different methods for estimating software maintenance effort in order to develop an algorithm for estimating software maintenance work based on object-oriented complexity derived from C&K metrics.

I. INTRODUCTION

Engineering and Computer Science Repairs are made to a system after it has been released so that defects may be fixed and functionality can be improved. After an issue has been fixed, maintenance is often utilised. Non-corrective procedures, on the other hand, account for more than 80% of repair labour, according to the study. Users may report bugs, and these complaints help to maintain the device's positive image by improving the user interface. The number of defects reported in the most recent reports has likewise increased by 21%. When Meir M. Lehman published his paper in 1969, he was the first to talk about the administration and development of computer software.
He was inspired to create The Laws of Lehman over a period of twenty years by his theory (Lehman 1997). In fact, the data demonstrates that maintenance is a dynamic process that changes over time (and software).

Lehman demonstrated the mutability of networks over time. Their complexity increases with time, necessitating techniques like code reconstitution to keep things simple. Lientz and Swanson's study from the late 1970s, which is well-known and often cited, showed the astronomically high maintenance costs.

About 75% of the repairs in the first two forms and 21% of the mistakes were corrected, according to the findings of the research study. Several subsequent investigations revealed a problem of a similar size. End users' involvement in the collection and evaluation of new needs has been shown in studies to be important.

Everything that goes wrong in gadget, develop and maintain stems from this. When software fails, it may mean lost revenue and lost chances for businesses. That's why computer service is so critical.

Maintenance has been categorized in three sections:

1. If an error is found after the code has been made publicly functional, it will be rectified. Corrective maintenance is when a developer fixes a compilation problem by changing a Java function.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 1, January 2017

2. Adequate maintenance requires the equipment to be re-tuned to perform in new environment, such as new network or operating system architecture. Adaptive services, such as this one, correct a Java method that only works with BEA WebLogic and not IBM Websphere.
3. Perfect maintenance includes any changes that make it possible for the programme to satisfy the same standards in a better manner. By just modifying the coding, developers may make a gadget simpler to maintain or operate.

AME = 0 because annual maintenance effort is estimated at 15 person-months per 100 machines. It's possible that this software project will take 15 months to complete each year.

II. C&K METRICS

C&K METRICS Six metrics are calculated for each class: WMC,DIT,CBO, NOC, LCOM, and RFC. The Booch's recommended OOD method has three stages that solve the following issues:

1. Class identification (WMC,NOC, DIT)
2. Class Semantics (LCOM,RFC,WMC)
3. Relationships among classes (RFC,CBO)

All of the courses that make up a single class are listed here. In other words, it means that certain classes' methods need or use other components. A class's DIT history shows how many other courses have come after it. C&K recommend that you follow their findings after looking at an example of depth. The LCOM cohesiveness is represented by an arbitrary number of paired approaches. The C&K number, which denotes the next level of class's subclasses, must be used to specify NOC. The RFC is a component of the feedback team. C&K builds feedback group as part of the methodology business to assist a class via linked groups. The World Media Conference (WMC) shows the level of technological complexity.

NOC: OOPS-based design categorizes things in this way. This is a collection of subclasses that are directly inherited from a super class in the class tree.

WMC: It's a mechanism based on objects. This is referred to as a function inner class count.

RFC: Each time a class object gets new information, a methodology group is started. A class's solution is in this collection of approaches.

Depth of inheritance tree (DIT): A class's best and most usual approach to its parent class becomes its best and most customary legacy. The DIT's superclass is empty.

Coupling between object classes (CBO): We may infer a link to RFC from this. This number incorporates categories that are subsets of a larger grouping of related categories.
Lack of cohesion methods (LCOM): Cohesion was taken into consideration as a measure when there were many unlinked method pairs in a class.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 1, January 2017

III. OBJECT ORIENTED METRICS AND SOFTWARE RELIABILITY

The link between OO metrics and software quality has been experimentally proven in several research. For forecasting or estimating fault-prone classes or system dependability, the chosen literature comprises prediction and estimate methods based on operational metrics (OO metrics).

Object-oriented software reliability models were studied by Sherry A.M., et. al., who suggested a novel model in which the number of starting parameters is an essential parameter of the dependability model. He discovered a link between the number of initial flaws in an object and several OOPS measures. Software metrics are being used by NASA projects in collaboration with SATC (the Software Assurance Technology Centre) to enhance the quality and dependability of software products, according to Rosenberg Linda and colleagues. When quality is present, reliability follows as a by-product. Reliability may be quantified. Early usage of metrics may help identify and rectify requirement flaws and ensure product dependability.

IV. SOFTWARE METRICS

This is a term used to measure the items i.e. software product, software process, person involved in software production or an organization such as data processing department.

Broadly software metrics are categorized into two parts:

1. Product Metrics

The software's qualities are measured with the help of product metrics. Metrics for products include things like dependability, functionality, performance, usability, cost, size, complexity, and style.

2. Process Metrics

They are used to assess efficiency of software development process. Measures for the process include the cost, effort, progress, and reuse metrics. It aids in estimating the eventual system size and assessing if a project is progressing as planned.

V. LITERATURE REVIEW

Numerous studies have been conducted in the area of estimating software effort. This section summarises some of the most recent findings in the field.

Mohamed Abdullahi Ali et al. presented their software component quality model in 2019. New applications are built by merging or replacing existing software components (CBSD). Reusing components will improve product quality while also speeding up software development. Computer-based systems design (CBSD) models are used to describe which modules do what.

Research conducted by Dr. Arvinder Kaur and colleagues [2] in 2010 looked at software maintenance prediction using computers. Maintenance effort has a non-linear relationship to object-oriented metrics. As a result, advanced technology must be developed and used to build models that anticipate programme maintenance requirements. ANN, Fuzzy Infer Systems, and other soft computing techniques are examined and compared in this article in order to better prepare computer maintenance tasks.

In 2015, Reenu Rani et.al[3] used open source software maintenance charges to reduce costs. In today's changing world, evaluating computer maintenance expenses becomes more crucial for adapting. The term "software maintenance expenditures" refers to the costs associated with maintaining software after its initial deployment.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 1, January 2017

Because maintenance costs might often exceed the cost of redeveloping the programme, the study's primary objective is to evaluate sustainability needs.

Mamta Punia et al. [4] introduced methods for software maintenance prediction calculation in 2014. Reusing old components while creating new software systems is possible with CBSE. This leads to better software quality as well as more efficient software development. Once the application has been installed on the client's site, there is a big issue with CBS service that must be addressed and remedied. For programmes like Very Decent, Good, Bad, and Second-Highest Rating one computational strategy is described in this article to automatically produce maintenance criteria such as Very Decent, Good, Bad, and Very Poor (the worst rating).

Kiran Narang et al[5] proposed the DRCE maintainability model in 2019 for soft computing technologies and CBS. Component-dependent computing requires efficient software maintenance, although doing so is difficult. Many maintenance solutions have been proposed by academics to minimise maintenance costs, improve programme uniformity, and extend device lifespans. Incorporating this strategy would allow software developers to construct applications that are simpler to administer in the long run. The article's approach emphasises four important maintenance requirements. Furthermore, their behaviours have an impact on system upkeep as a whole.
Bruce Edson and colleagues proposed Software Reliability and Maintenance for the first time in 1996. The Network Engineering Framework ensures software availability, maintenance, and stability at all time points in the software life cycle. Project computers' post-deployed service mechanisms were administered and executed using Product Reliability, availability, a stability and maintenance engineering instrument developed by the Group of Air Force Material Command Space Systems (PRV).

Charu Singh and colleagues [7] developed the Fuzzy Logic Technique in 2014 to assess software reusability. Developing IT applications requires the use of soft computing techniques. Fuzzy logic systems, neural network models, and genetic algorithm techniques are examples of cutting-edge innovations. In order to analyse a device's reusability, maintenance, and comprehension, fuzzy logic and neural networks are also employed.
Parallel computing was used in combination with a hybrid neural network/fluid-based logic method by Lov Kumar etal [8]. Object-oriented software is currently produced and used by the great majority of technology companies for all of these reasons and more. As a result, a high standard of programming is becoming more crucial. A neural network and a fluffy logical technique were used to provide ten object-oriented static source code metrics for feedback in this research. Data from two commercial software devices, UIMS and QUES, is used to measure the method's upkeep.

It was stated in 2016 by Shivani Kundu et.al [9] that they had completed a thorough literature review on software maintenance management approaches. A good software system's consistency is largely determined by how well it is maintained. High-quality programmes make advantage of component-based designs and object-focused approaches to make maintenance easier.
In 2012[12], Dubey et al.[10] presented an ambiguous technique for object-oriented software system maintenance assessments. Effective information systems are becoming more important as time passes. The importance of software maintenance in the development of effective systems cannot be overstated. According to recent technological breakthroughs, most computers now build high-quality software utilising object-oriented methodologies.
Researchers Sharawat and coworkers [11] published their findings in 2012. Neural networks may be used to forecast software maintainability. Using the Maintainability Index, which is a hybrid statistic that combines numerous

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 1, January 2017

traditional source code indications into a single long-term value, software maintenance is made simpler. Artificial intelligence (AI) neural networks are built using Li-Henry data, which forecasts MI. Choose from trainlm, trainrp, or another training technique to suit your needs.
Researchers Zavvar et.al [12] used the Adaptive Fuzzy Network to evaluate software maintenance in 2015. When we talk about "software repair," we're talking about the techniques used to fix the programme after it has been delivered. The proposed technique's root mean square error was 0.34331 in this study, and it was used to assess the maintenance program's overall capability after it underwent testing by an adaptive fuzzy neural network. As a consequence of these findings, the adaptive fugitious neural approach may be used to evaluate software output for maintenance purposes.

In 2015, Mishra et al [13] published a study. The art of prediction in OO software Technology for an affordable and adaptable fuzzy network-based network today, software developers face one of their most difficult tasks yet: determining how complicated, expensive, and time-consuming a programme is. For software managers, this has important consequences since the estimations above and below has a considerable effect on the profitability of software firms. Data on the maintenance of two commercial software packages was used in the research for this study. The study's main focus is on the independent variable, shift.Nine distinct measurements based on relationships between various parameters make up the independent variables.

Software prediction clustering was proposed by Yuan et al [14] in 2000. Due to increasing software dependability requirements, it is necessary to create software quality prediction technologies that can provide accurate predictions. Using fading subtractive clustering and module-order modelling for software output evaluation, the author proposes a modelling method. Modules with an acceptable accuracy of customer-discovered faults may be identified before publication, according to the case study, using subtractive fading clustering and modular order modelling.
Examining software maintenance via research N. Barbosa et al.[15] carried out research in 2013 that was used by C&K to develop their metrics. As soon as possible preventative or corrective action may be taken in order to guarantee that essential services are preserved or deployed in more challenging regions or future stages, income will be generated at different project levels via fast and early software maintenance assessment. In order to assess if the unique programme is being maintained, Chidamber & Kemerer's measures will be statistically analysed (C&K).
A study in 2006 projected that applications employing multivariate adaptive regression splines would achieve their goals, and that prediction has since been corroborated. The findings show that MARS has improved its ability to predict maintenance needs.

Tahir et al.[17] developed an AOP method in 2010 for collecting software dynamic maintenance metrics. Because of this, app developers are concerned about the quality of their products. Performance metrics are often used in quantitative software quality assessments. Static metrics are often used despite the fact that dynamic measures may show software quality problems more clearly. This is primarily responsible for the technological challenges involved with dynamic metrical selection.
Static dependability and maintenance measures were studied by J. Ludwig et al[18] in 2018. A variety of static code metrics related to software quality and maintainability are discussed in this article, as well as the most common sources of technical debt.
It was suggested in 2014 that software maintenance metrics be automatically gathered by J. Ostberg et.al [19].
Masmali et.al [20] proposed the Comprehensive Complexity Metrics Complexity System in 2020. Measuring programmed complexity is critical for software lifecycle management and maintenance. While software code complexity may be measured using well-developed and rigorous benchmarks, software ideas are nevertheless

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 1, January 2017

difficult to pin down because of the inherent fuzziness. There are also no specific recommendations for software designers to choose solutions that reduce the design's flexibility, increase the design's understandability, and improve software maintenance. This article provides a quantitative and definitive criteria for evaluating the difficulty of programmer architecture that uses an independent language approach. UML Class Diagrams and UML State Machines, a book written for programmers, highlights two important points. The focus of the method is on research and the reciprocal connection between the many components of the final design. More UML notifications could be supported with the current technology.

VI. PROBLEM STATEMENT

C&K metrics optimization for object-oriented testing must be presented as a technique of effort evaluation. Object-oriented modules must be studied to prioritise the effort assessment in light of the C&K metrics characteristics. There must be a research. This method of estimating maintenance work should be possible using subjectivity-oriented programming.

Using the C&K test metrics as a guide, this section suggests a technique for estimating work. The job that was suggested evolved over time. The whole procedure is well described. A flow diagram was used to describe the project's requirements. C&K-metrics features are used to evaluate effort for object-oriented modules.

The proposed method is suitable for giving emphasis to the effort in object-driven programming. A flow chart depicts the proposed approach.

VII. FUTURE SCOPE

Object-oriented software effort estimates will benefit greatly from the proposed investigation. OOPS is often used in software development. As a consequence, there is a growing need for a priority setting method that is sufficient and efficient during anticipated effort. C&K Metrics support may prioritise objective-based programming modules like Java, C++, and C. These studies might be critical in figuring out if the module's weighted average exhibited the same position across instances

REFFERENCES

1. Mohamed Abdullahi Ali, Ng Keng Yap, “Software Component Quality Model”, International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958,Volume-9, Issue-1, October 2019
2. Kaur, Arvinder, Kamaldeep Kaur, and Ruchika Malhotra. "Soft computing approaches for prediction of software maintenance effort." International Journal of Computer Applications 1, no. 16 (2010): 69-75.
3. Rani, Reenu, Dipen Saini, and Amandeep Kaur Cheema. "OPEN SOURCE SOFTWARE MAINTENANCE COST EVALUATION USING MAINTAINABILITY INDEX AND CODE METRICS."
4. Punia, Mamta, and Amandeep Kaur. "Software maintainability prediction using soft computing techniques." IJISET 1, no. 9 (2014): 431-442.
5. Narang, Kiran, Puneet Goswami, and K. Ram Kumar. "Maintainability Configuration for Component-Based Systems Using Fuzzy Approach." In Computational Methods and Data Engineering, pp. 243-258. Springer, Singapore, 2021.
6. Edson, Bruce, Bill Hansen, and Pat Larter. "Software reliability, availability, and maintainability engineering system (SOFT-RAMES)." In Proceedings of 1996 Annual Reliability and Maintainability Symposium, pp. 306-311. IEEE, 1996.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 1, January 2017

7. Singh, Charu, Amrendra Pratap, and Abhishek Singhal. "An estimation of software reusability using fuzzy logic technique." In 2014 International Conference on Signal Propagation and Computer Technology (ICSPCT 2014), pp. 250-256. IEEE, 2014.
8. Kumar, Lov, and Santanu Ku Rath. "Software maintainability prediction using hybrid neural network and fuzzy logic approach with parallel computing concept." International Journal of System Assurance Engineering and Management 8, no. 2 (2017): 1487-1502.
9. Kundu, Shivani, and Kirti Tyagi. "Effort estimation of software maintainability using soft computing techniques: a critical literature survey." World Applied Sciences Journal 34, no. 6 (2016): 733-742.
10. Dubey, Sanjay Kumar, and Ajay Rana. "A fuzzy approach for evaluation of maintainability of object oriented software system." International Journal of Computer Applications 49, no. 21 (2012).
11. Sharawat, Mr Sandeep. "Software maintainability prediction using neural networks." environment 3, no. 5 (2012): 750-755.
12. Zavvar, Mohammad, and Farhad Ramezani. "Measuring of software maintainability using adaptive fuzzy neural network." International Journal of Modern Education & Computer Science 7, no. 10 (2015): 27-32.
13. Mishra, Swati, and Arun Sharma. "Maintainability prediction of object oriented software by using adaptive network based fuzzy system technique." International Journal of Computer Applications 119, no. 9 (2015).
14. Yuan, Xiaohong, Taghi M. Khoshgoftaar, Edward B. Allen, and K. Ganesan. "An application of fuzzy clustering to software quality prediction." In Proceedings 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology, pp. 85-90. IEEE, 2000.
15. Barbosa, Nelson, and Kechi Hirama. "Assessment of software maintainability evolution using C&K metrics." IEEE Latin America Transactions 11, no. 5 (2013): 1232-1237.
16. Zhou, Yuming, and Hareton Leung. "Predicting object-oriented software maintainability using multivariate adaptive regression splines." Journal of systems and software 80, no. 8 (2007): 1349-1361.
17. Tahir, Amjed, and Rodina Ahmad. "An AOP-based approach for collecting software maintainability dynamic metrics." In 2010 Second International Conference on Computer Research and Development, pp. 168-172. IEEE, 2010.
18. Ludwig, Jeremy, Steven Xu, and Frederick Webber. "Static software metrics for reliability and maintainability." In 2018 IEEE/ACM International Conference on Technical Debt (TechDebt), pp. 53-54. IEEE, 2018.
19. Ostberg, Jan-Peter, and Stefan Wagner. "On automatically collectable metrics for software maintainability evaluation." In 2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement, pp. 32-37. IEEE, 2014.
20. Masmali, Omar, and Omar Badreddin. "Comprehensive Model-Driven Complexity Metrics for Software Systems." In 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), pp. 674-675. IEEE, 2020.
21. Shepperd, M. J. "System architecture metrics for controlling software maintainability." In IEE Colloquium on Software Metrics, pp. 4-1. IET, 1990.
22. Genero, Marcela, Mario Piattini, Esperanza Manso, and Giovanni Cantone. "Building UML class diagram maintainability prediction models based on early metrics." In Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No. 03EX717), pp. 263-275. IEEE, 2004.
23. Garcia, Felix, Francisco Ruiz, and Corrado Aaron Visaggio. "A proposal and empirical validation of metrics to evaluate the maintainability of software process models." In 2006 IEEE Instrumentation and Measurement Technology Conference Proceedings, pp. 1093-1097. IEEE, 2006.

ISSN(Online) : 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Website: www.ijirset.com

Vol. 6, Issue 1, January 2017

24. Abílio, Ramon, Pedro Teles, Heitor Costa, and Eduardo Figueiredo. "A systematic review of contemporary metrics for software maintainability." In 2012 Sixth Brazilian Symposium on Software Components, Architectures and Reuse, pp. 130-139. IEEE, 2012.
25. Jin, Cong, and Jin-An Liu. "Applications of support vector mathine and unsupervised learning for predicting maintainability using object-oriented metrics." In 2010 Second International Conference on Multimedia and Information Technology, vol. 1, pp. 24-27. IEEE, 2010.
26. Oman, Paul, and Jack Hagemeister. "Metrics for assessing a software system's maintainability." In Proceedings Conference on Software Maintenance 1992, pp. 337-338. IEEE Computer Society, 1992.
27. Stoddard, Robert W. "An extension of goal-question-metric paradigm for software reliability." In Proceedings of 1996 Annual Reliability and Maintainability Symposium, pp. 156-162. IEEE, 1996.
28. A. Jatain and G. Sharma, "A Systematic Review of Techniques for Test Case Prioritization," Int. J. Comput. Appl., vol. 68, no. 2, pp. 38–42, 2013, doi: 10.5120/11554-6833.
29. G. Duggal, "Understanding regression testing mechanisms," 2nd Natl. Conf. Challenges Oppor. Inf. Technol., no. 1, 2008.
30. R. Huang, W. Zong, D. Towey, Y. Zhou, and J. Chen, "An empirical examination of abstract test case prioritization mechanisms," Proc. - 2017 IEEE/ACM 39th Int. Conf. Softw. Eng. Companion, ICSE-C 2017, pp. 141–143, 2017, doi: 10.1109/ICSE-C.2017.105.
31. S. Nayak, C. Kumar, and S. Tripathi, "Effectiveness of prioritization of test cases based on Faults," 2016 3rd Int. Conf. Recent Adv. Inf. Technol. RAIT 2016, pp. 657–662, 2016, doi: 10.1109/RAIT.2016.7507977.
32. W. Fu, H. Yu, G. Fan, and X. Ji, "Test case prioritization approach to improving effectiveness of fault localization," Proc. - 2016 Int. Conf. Softw. Anal. Test. Evol. SATE 2016, pp. 60–65, 2016, doi: 10.1109/SATE.2016.17.
33. Y. Wang, X. Zhao, and X. Ding, "An effective test case prioritization method based on fault severity," Proc. IEEE Int. Conf. Softw. Eng. Serv. Sci. ICSESS, vol. 2015-Novem, pp. 737–741, 2015, doi: 10.1109/ICSESS.2015.7339162.
34. A. Gonzalez-Sanchez, R. Abreu, H. G. Gross, and A. J. C. Van Gemund, "Prioritizing tests for fault localization through ambiguity group reduction," 2011 26th IEEE/ACM Int. Conf. Autom. Softw. Eng. ASE 2011, Proc., pp. 83–92, 2011, doi: 10.1109/ASE.2011.6100153.
35. Santanu Kr. Misra, "Assessment of Object Oriented Metrics for Software Reliability," International Journal of Engineering Research & Technology (IJERT), ISSN: 2278-0181, Vol. 4 Issue 01January-2015.